IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 21, 2019, accepted February 10, 2019, date of publication February 15, 2019, date of current version March 5, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2899653

eTDP: Enhanced Topology Discovery Protocol for Software-Defined Networks

LEONARDO OCHOA-ADAY, (Student Member, IEEE), CRISTINA CERVELLÓ-PASTOR, (Member, IEEE), AND ADRIANA FERNÁNDEZ-FERNÁNDEZ, (Student Member, IEEE)

Department of Network Engineering, Universitat Politècnica de Catalunya, 08860 Barcelona, Spain

Corresponding author: Leonardo Ochoa-Aday (leonardo.ochoa@entel.upc.edu)

This work was supported by the Ministerio de Economía y Competitividad of the Spanish Government under Project TEC2016-76795-C6-1-R and AEI/FEDER, UE.

ABSTRACT Discovering network elements in a dynamic and optimized manner and being able to contend with ever-growing traffic is a key requirement for current networking environments. In software-defined networks (SDNs), the controller collects the topology information from the data plane and maintains an abstract view of the entire network, which is crucial for the proper functioning of applications and network services. However, there is still the need for an enhanced protocol for automatic discovery and mechanisms of autoconfiguration of network elements according to new policies and business requirements. To overcome this challenge, this paper presents a novel protocol that, unlike existing approaches, enables a distributed layer-2 discovery without the need for previous network configurations or controller knowledge of the network. By using this mechanism, the SDN controller can discover the network view without incurring scalability issues, while taking advantage of the shortest control paths toward each switch. The obtained results show that our enhanced protocol is efficient in terms of time and message load over a wide range of generated networks and outperforms the state-of-the-art techniques.

INDEX TERMS Network management, protocols, software-defined networks, topology discovery.

I. INTRODUCTION

The evolution of information and communication technologies (e.g. cloud computing, the Internet of Things (IoT) and 5G, among others) has enabled a large market of new applications and network services for a massive number of users connected to the Internet. Achieving high programmability while decreasing complexity and costs has become an essential aim of networking research due to the ever-increasing pressure generated by these applications and services. However, achieving these goals is an almost impossible task using traditional IP networks.

To be able to address these high demands from users, network operators will require emerging solutions to effectively manage their network resources in a dynamic and flexible manner. In addition, in order to deploy high-level policies in traditional networks, operators need to configure each element of the network. This often occurs via specific, low-level commands from manufacturers because the plane that determines how to manage traffic (the control plane) and the plane that forwards traffic in accordance with the decisions of the control plane (the forwarding plane) are vertically integrated into a single network device.

The term ''programmable networks'' is usually employed to describe the desired future of networking. In essence, a network is said to be programmable if the behavior of its network devices and its traffic control are managed by software that operates independently from the network's physical infrastructure. Moving from closed, proprietary-based computer hardware to software-oriented (and thus programmable) networks provides the opportunity for networking innovation, making it possible and more straightforward to evolve network capabilities and deploy new services.

Commonly used as a synonym for programmable networks, Software-Defined Networking (SDN) is an emerging network architecture intended to address the increasing needs of data centers and campus networks, as well as the requirements of service providers in carrier environments [1].

The associate editor coordinating the review of this manuscript and approving it for publication was Byung-Seo Kim.

 This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/

In contrast to traditional IP networks, SDN enables researchers and network administrators to design highly nuanced network control that better corresponds to the current and dynamic demands of users. The novel network design proposed by SDN, i.e. decoupling the control plane from the underlying forwarding plane, evolves traditional network infrastructures from configurable to programmable [2]. Consequently, it enables the introduction of new protocols and traffic management models with logically centralized control policies across multi-vendor and multi-layer networks.

In SDN, the controller maintains a holistic view of the network through the topology discovery service. This topological knowledge is crucial for the correct operation of other internal controller services such as hosts tracking and network configuration, as well as for other network applications (e.g. traffic engineering, network monitoring, attack detection and routing protocol, among others) that run on top of the considered architecture [3].

II. TOPOLOGY DISCOVERY SERVICE IN SDN

Network topology discovery is a description of the physical structure of the network. This description contains information about the connectivity and link capacity between the network devices at the lowest level (i.e. the data plane layer). This physical infrastructure represents the overall resources of the network.

In general, topology discovery is highly important in several computer network areas such as routing, resource allocation and configuration, Quality of Service (QoS), network management, diagnosis and fault recovery, among others. For this reason, discovering the current topology of a network is a compulsory task for every network operator. Moreover, collecting this real-time information efficiently and automatically is critical for significant networking problems such as enhancing network connectivity and resolving network congestion. In order to improve the performance of these network services, preserving an accurate view of the network topology at all times is also an important task.

However, maintaining a comprehensive view of large networks generates a considerable amount of state information from the forwarding plane [4]. Furthermore, a substantial volume of state information represents considerable pressure for the central controller and, as a consequence, scalability issues might appear [5]. Schemes in which the controller periodically sends multiple packets to every forwarding node in the network could overload the controller's performance.

Although discovering the network topology is an essential service of the SDN controller, at the time this writing no official standard defines the topology discovery mechanism in SDN [6]. Due to this issue, most current controllers (e.g. POX [7], Ryu [8], among others) implement a topology discovery mechanism based on the popular southbound protocol OpenFlow.

This discovery mechanism, referred to as OpenFlow Discovery Protocol (OFDP) [9], [10], implements a topology discovery technique that uses the frame format defined by the Link Layer Discovery Protocol (LLDP) [11]. Except for the frame format, OFDP does not have much in common with LLDP. Basically, OFDP is based on packet-out and packet-in messages between the controller and switches.

In addition to LLDP, some SDN controllers such as OpenDayLight [12] and ONOS [13] implement the Broadcast Domain Discovery Protocol (BDDP) to discover network links when OpenFlow switches are separated by a non-OpenFlow switch [14]. This protocol has the same structure that LLDP packets but instead of using a multicast address, the destination MAC address field contains a broadcast address.

In order to discover the topology under OFDP, switches require two major previous configurations. Firstly, every switch has initially programmed the IP address and TCP port of its controller to establish a connection as soon as the device is turned on. Secondly, switches have preinstalled flow rules to route directly to the controller via a packet-in message, any LLDP packet received from another switch.

Using the IP address and TCP port programmed in advance, the switch searches for its controller in the network and attempts to establish a secure and encrypted connection through a Transport Layer Security (TLS) session. The controller as part of this initial handshake sends a feature request message to the switch, which responds with a feature reply message. With this message the switch informs the controller about relevant parameters for the network discovery such as the switch ID, a list of active ports with their corresponding MAC addresses, among others.

In essence, under OpenFlow, switches are discovered and added to the network view in the initial handshaking process. Consequently, with OFDP the topology discovery is reduced to discover the inter-connected links between the switches. Moreover, sending messages periodically from the controller to each OpenFlow switch increases the network traffic and latency between the control plane and the forwarding plane, and can also lead to network limitations and outages [15], [16].

An efficient and straightforward mechanism for topology discovery in large-scale SDN could be achieved by dividing the entire process into phases and distributing the discovery functions hierarchically between the network nodes [17]–[19]. This allows for obtaining the network graph as quickly as possible without incurring scalability issues. Following this logic, this paper presents a novel topology discovery protocol called Enhanced Topology Discovery Protocol (eTDP).

Different from previous work, eTDP is implemented in each switch through a basic software agent that executes simple decisions. Thus, this approach provides a distributed solution, as the nodes that support the network protocol perform the topology discovery process. In essence, this contribution is designed to obtain an abstract view in large-scale networks. It minimizes both the time and the number of required packets, while simultaneously decreasing the

overload in controller performance to reinforce the current topology discovery service in SDN.

The remainder of this chapter is organized as follows. In Section III, we first present the literature review, which contains research relevant to the paper scope. Then, we explain the primary considerations of our approach and fully describe the proposed protocol in Section IV. In Section V, we present the simulations conducted and analyze the achieved results. Finally, in Section VI the main conclusions of this work are outlined.

III. LITERATURE REVIEW

In this section, we first outline a general survey of OpenFlow-based discovery mechanisms in programmable networks. After this, we provide a brief description of the proposals in existing literature that have considered applying non-OpenFlow solutions to the topology discovery problem in SDN.

A. OpenFlow-BASED APPROACHES

Currently, in SDN infrastructures, after OpenFlow compliant switches are turned on the SDN controller establishes an initial control connection with each forwarding device [20], [21]. This initial handshake is used by the SDN controller to request capabilities from the switches, such as configuration information, the number of active interfaces (i.e. network ports), corresponding MAC addresses, etc [20].

After this, the controller initiates the de-facto topology discovery in SDN, called OFDP. For this protocol, the SDN controller begins by sending LLDP frames encapsulated in packet-out messages to each active interface on each OpenFlow switch in the network. By default, after receiving an LLDP packet from ports other than the controller port, each active switch must send a packet-in message that contains the received LLDP to the controller.

Both OpenFlow packet-in and packet-out messages are essential for current topology discovery mechanisms. The number of packet-out messages that an OpenFlow controller must send is equal to the total number of ports in the network. Meanwhile, the total packet-in messages it receives is twice the number of active links in the network, as there is one packet for each direction [14]. Therefore, due to OFDP, the controller load is determined by the number of packet-out and packet-in messages that the controller must process. The related works discussed below propose improving the efficiency of the OFDP mechanism based on the reduction of packet-out messages sent from the SDN controller to OpenFlow switches.

Pakzad *et al.* [22], [23] evaluate the efficiency of the OFDP mechanism implemented by current SDN controllers. The authors propose simple and practical modifications to reduce controller overhead during the topology discovery procedure and implement an OFDPv2 based on the ability of OpenFlow switches to rewrite packet headers. As a result, the number of packet-out messages sent by the controller can be reduced to only one message per OpenFlow switch. In [22], after implementing the improved approach using a POX controller [7] and the Mininet emulator [24], results showed a reduction in the controller overload of up to 45%. Testing the proposed modification in a specific topology in the OFELIA SDN testbed [23] showed a reduction in controller overload of up to 40%, while the physical topology is presented as in the legacy OFDP mechanism.

Hasan and Othman [25] revisit the current OpenFlow-based topology discovery protocols, taking into account the effects of retransmitting the discovery packets until the SDN controller identifies the entire map of the network. To that end, the authors re-implemented the OFDPv2 proposed in [22] and [23] and compared it with the standard OFDP protocol. Experimental simulations revealed different patterns when retransmission of OpenFlow packets is taken into consideration. Specifically, although retransmission doubles the number of required packets, their implementation outperforms the basic OFDP in terms of bandwidth consumption.

The Tree Exploration Discovery Protocol (TEDP) is proposed in [26]. This protocol gathers topology information and simultaneously provides the shortest paths among forwarding devices without adding additional messages, as compared to current OFDP. After describing two possible implementations for TEDP, the authors also list some desired features that should ideally appear in future SDN platforms according to their research.

Azzouni *et al.* [27] introduce Secure and Efficient Topology Discovery Protocol (sOFTDP) as a novel and efficient alternative protocol to the current OFDP. This proposed protocol requires minimal changes to OpenFlow switch design and eliminates serious vulnerabilities in the topology discovery process. The authors implemented this solution as a topology discovery module in a Floodlight controller [28] and confirmed that it reduces the topology discovery time by several orders of magnitude in proof of concept experiments.

Taking into account multi-controller environments, Huang *et al.* [29] designed and implemented an automatic network topology discovery mechanism across various OpenFlow domains. To do this, Gude *et al.* [30] propose modifications to the discovery, topology and LAVI modules within the NOX controller. Furthermore, modifications to the display user interface of the ENVI module were made in order to connect all deployed NOXs and obtain the status of their OpenFlow switches. The authors implemented their solution in a large-scale OpenFlow testbed and the experiment displayed the entire topology across various domains within the same user interface

In [22], [23], [25]–[27], and [29], the controller needs, as previous knowledge, the IP address and the list of active ports of each SDN switch in the network. This knowledge is obtained through the establishment of an initial handshake between the controller and forwarding devices. In order to establish this initial switch-controller connection, it is assumed that each switch has programmed the IP address and

TCP port number of its controller. In addition, if the network topology changes, the OFDP mechanism can generate an excessive quantity of state messages. As a result, the network can experience service limitations and outages.

In contrast to this, we propose a topology discovery protocol to discover the network nodes that does not rely on offline device configurations and the exchange of topological information between the SDN controller and the switches over a secure connection. Based on this procedure, our protocol enables controllers to be connected in already deployed networks without the need of human intervention to configure each forwarding device. This feature provides a suitable approach for quick installation and plug-and-play controller/switch provisioning, which is an important challenge in the operation and management of current networks.

B. NON-OpenFlow APPROACHES

Most topology discovery mechanisms proposed thus far for SDN have focused on improving the current OFDP mechanism. However, other possibilities have also been studied due to the aforementioned limitations of OpenFlow-based solutions. In this study we have also analyzed some research contributions that discover the network topology in SDN using non-OpenFlow mechanisms.

Tarnaras *et al.* [31], [32] proposed an automatic topology discovery algorithm, which takes into account a better usage of the LLDP protocol. This protocol is used locally on the data plane of switches for updating their local neighbors table periodically. The authors obtained the topology map using the Forwarding and Control Element Separation (ForCES) framework for extracting LLDP data directly from the network devices. This procedure automatically reports any change in the physical topology to the controller as a triggered event. After implementing the proposed algorithm, the simulation results showed that the average time to discover a new switch (i.e. 12 ms) during the topology discovery process is 90% less than the OpenFlow-based solution (i.e. 100 ms).

Likewise, Jiménez *et al.* proposed the SDN Resource Discovery Protocol (SDN-RDP) as an alternative to distribute the management of the network state among several SDN controllers. Each controller discovers a portion of the network topology in order to ensure the distribution of node management and also simplify the protocol resolution. The described mechanism is asynchronous, lacks a global initialization process and does not require previous knowledge of the network. Based on simulation results, the proposed protocol achieves an efficient reduction of the controller overload.

In [34], a similar distributed operation was briefly introduced to support self-healing properties in SDN. However, this related work lacks of crucial features for protocol implementation such as message types and dataframes structure. In addition to providing further detail about the protocol design, in the present work a deeper evaluation is performed taking into account the impact of the control traffic on the network due to the topology discovery mechanism.

Choi *et al.* [35] proposed a topology discovery protocol called Generalized TOPology (G-TOP) for a stateful Path Computation Element (PCE). This protocol allows the PCE to automatically construct the network topology as a controller without using a distributed routing protocol (e.g. Open Short Path First-Traffic Engineering (OSPF-TE)). The proposed protocol proactively collects the topology information from the switch and reactively updates the topology changes through an out-of-band control channel. After implementing the proposed protocol, the total time for updating the topology was about 10 ms for the testbed system described in the paper. However, Jalili *et al.* [36] used out-of-band management, which might not be possible to deploy in some real, large-scale scenarios.

Although these mechanisms [22], [23], [25]–[27], [29], [31]–[33] are able to discover the network topology in SDN, they all require previous configurations within the network devices. Consequently, there is still a lack of more flexible topology discovery mechanisms in SDN based on layer 2 techniques. Our research is therefore intended to solve this issue.

A summary of the discussed topology discovery mechanisms is presented in Table 1. Each row in the table refers to a different approach. Meanwhile, columns refer to a particular feature: proposal description, considered technique and the requirement that forwarding devices must have configured the controllers' IP addresses. This paper is also included at the end of the table.

IV. ENHANCED TOPOLOGY DISCOVERY PROTOCOL

Although SDN provides a flexible architecture by centralizing network intelligence, the controller intervention in those control tasks that only require local switch knowledge is not always ideal. The execution of such tasks (e.g. neighbor discovery) can be delegated to the forwarding devices, which can gather the corresponding information and send it to the controller. In this way, the controller remains responsible for performing those tasks that require a global network view and centralized control.

A. PROGRAMMABLE NETWORK INFRASTRUCTURE

The proposed solution can be implemented in a network system following the recent design proposed by the SDN paradigm [37]. The overall system architecture for the proposed solution is shown in Fig. 1. This network system embraces network control decoupled from forwarding devices and leverages SDN controllers to provide an abstract view of the entire network.

This proposal can be deployed in a network domain with multiple SDN controllers through the use of a software agent (e.g. eTDP client) running in each network device. As a result, based on the topology information sent by switches each SDN controller discovers and maintains an accurate network view in the topology database. The stored information is critical for the proper operation of other controller services and supported network applications (e.g. traffic engineering,

TABLE 1. Comparison between the proposed approach and other state-of-the-art solutions.

Proposal	Brief Description	Technique	Initial IP connection required?
[22], [23]	Modified OFDPv2 that reduces the number of packet-out messages, sent by the SDN controllers, to only one message per OpenFlow switch	OpenFlow-based	Yes
[25]	OFDPv2 implementation that takes into account the retransmission of OpenFlow packets until the SDN controller identifies the entire map of the network	OpenFlow-based	Yes
[26]	The proposed TEDP gathers topology information and simultaneously provides the shortest paths among forwarding devices without adding extra messages	OpenFlow-based	Yes
[27]	Implemented in a Floodlight controller, sOFTDP reduces the topology discovery time by several orders of magnitude in proof of concept experiments	OpenFlow-based	Yes
[29]	Automatic topology discovery mechanism across various OpenFlow domains, which is implemented using the NOX controller in a large-scale testbed	OpenFlow-based	Yes
[31], [32]	Automatic topology discovery algorithm based on the ForCES framework and periodically used by switches for updating their local neighbors table	Non-OpenFlow-based	Yes
[33]	SDN-RDP distributes the management of the network state among several SDN controllers while reducing the controller overload	Non-OpenFlow-based	Yes
[34]	Distributed topology discovery mechanism that supports self-healing properties to boost the control plane resilience in SDN	Non-OpenFlow-based	No
[35]	The G-TOP protocol allows a stateful PCE to construct the network topology automatically as a controller without using a distributed routing protocol	Non-OpenFlow-based	No
Current paper	Distributed layer 2 topology discovery protocol without requiring previous network configurations or controller knowledge of the network	Non-OpenFlow-based	No

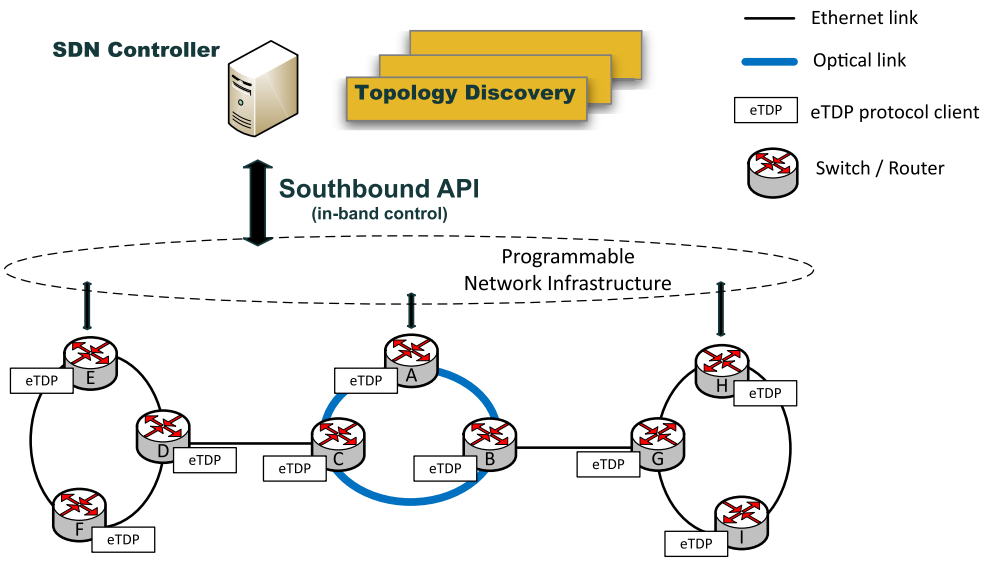

FIGURE 1. Overall system architecture for the proposed solution.

network telemetry and attack detection, among many others).

This proposal is deployed in the data plane through the use of a software agent (e.g. eTDP client) running in each network device. Administrators can install and activate these agents on the forwarding devices as needed whenever the network infrastructure grows.

In a network domain with multiple SDN controllers, each controller, based on the topology information sent by switches, discovers and maintains an accurate network view in the topology database. The stored information is critical for the proper operation of other controller services and supported network applications (e.g. traffic engineering, network telemetry and attack detection, among many others).

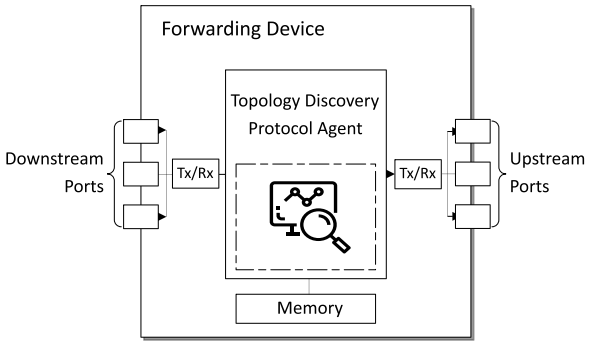

FIGURE 2. Schematic diagram of a forwarding network device.

As shown, the control plane can be interconnected with a plurality of network devices (e.g. traditional routers, virtual network devices, programmable switches, etc.) through different transmission media (e.g. fiber optic links, electrical links, wireless links or logic connections). Although in Fig. 1 SDN controllers use an in-band control scheme [38], this enhanced approach can also be implemented using out-of-band connections.

B. FORWARDING NETWORK DEVICE

The network devices may be any hardware-based (i.e. switch or router) or software-based (logical or virtualized) device configured to perform data forwarding functions according to the routes specified by the SDN controller. Fig. 2 presents a schematic diagram of a network device.

The Topology Discovery Protocol Agent is the component responsible for performing the proposed eTDP at each node, which can be implemented using an agent-oriented approach. These agents perform a local partial function of the entire discovery process while interacting autonomously. This capability of distributed operation allows the global topology discovery task to evolve in a scalable and effective way, without overburdening the SDN controller. Topology information retrieved by each node during the eTDP operation is temporarily stored in the device memory and periodically sent to the controller of the corresponding SDN domain.

C. CONTROL FRAMES DESCRIPTION

The eTDP communications are carried out using a standard network frame format for all data related to the protocol. This feature allows us to develop future extensions of the protocol while maintaining compatibility with previous versions. Moreover, the packets are encapsulated with their corresponding headers (i.e. MAC or IP) for transmissions over the network.

1) MESSAGE HEADER

Every message sent by the eTDP is encapsulated according to the same header structure shown in Fig. 3. Note that each tick mark represents a one-bit position in the frames and that the fields are transmitted from left to right.

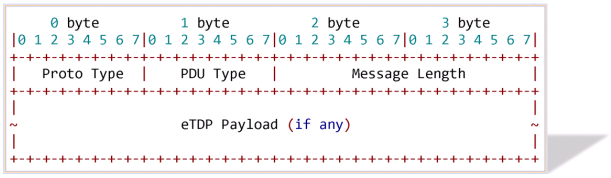

FIGURE 3. General structure of eTDP messages.

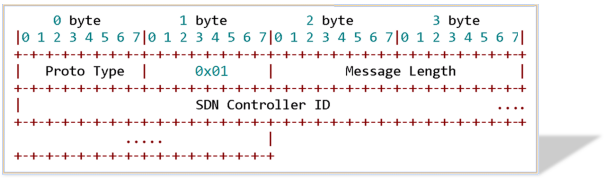

FIGURE 4. topoRequest message format.

These messages share a common header format, which allows a node to be able to accept or relay (if applicable) messages of different types. This feature supports fine-grained message forwarding using the powerful "match + action" abstraction of SDN. The definitions of each field included in the message headers are further described below:

1) *Proto Type*: Protocol type (8 bits). This field uses a specific hexadecimal number to denote the protocol type so that any switch that supports this protocol can easily identify eTDP messages in the network control frame.
2) *PDU Type*: Packet data unit type (8 bits). This field contains a value that specifies the type of message in the payload. For example, type 0x01 denotes a topoRequest frame, type 0x02 indicates an echoReply frame and type 0x03 corresponds to a topoReply frame.
3) *Message Length*: Message size (16 bits). This field indicates the message end in the byte stream, starting from the first byte of the header.

As illustrated in Fig. 3, the overall header size is 32 bits (i.e. 4 octets). Some field values (e.g. *PDU Type* and *Message Length*) used in this fixed structure depend on the kind of eTDP messages sent by the network nodes.

2) topoRequest

The topoRequest message is used by the SDN controller to initiate the topology discovery process in the network. Fig. 4 presents the message format of a topoRequest.

Besides the header, this simple message only carries the ID corresponding to the SDN controller that sends the topoRequest message. This is the manner in which each SDN controller announces its presence to every forwarding device active in the network.

1) *SDN Controller ID*: Controller identifier (48 bits). The node identifier used in the messages is the MAC address. If the network controller has more than one interface, it must choose the MAC address from one of its active interfaces. This field has the

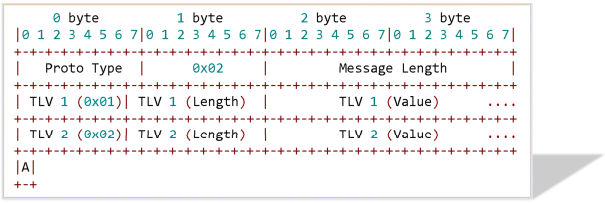

FIGURE 5. echoReply message format.

same value for every topoRequest message sent by the SDN controller.

3) echoReply

After receiving the topoRequest message, each network node should automatically reply with an echoReply message. This one-hop reply enables the exchange of local topology information between neighbors and the establishment of a hierarchical control tree rooted at the SDN controllers. In Fig. 5 the frame format of an echoReply message is presented. As shown, the value in the message header (i.e. *PDU Type*), has changed to type 0x02 for indicating the echoReply message.

This message format was inspired by the use of Type-Length-Value (TLV) structures for the exchange of local neighbor information. Type-Length-Value structures have been widely exploited by several existing standardized protocols such as LLDP [11], Intermediate System to Intermediate System (IS-IS) [39] and Remote Authentication Dial-In User Service (RADIUS) [40], among others. In this proposal we utilized TLV as an efficient method for transmitting different kinds of topology data inside the message body.

While the TLV type and length fields occupy the first two octets of the TLV format, the value field may have a fixed or variable size. In addition, it may include different types of information, containing either binary or alphanumeric data, which is specified using the associated subtype identifiers (e.g. port component, MAC or IP address, interface name, locally assigned identifiers, etc.).

Table 2 describes the standard TLVs supported by this protocol. Complementary TLVs can also be defined to enable protocol extensions. Specifically, in the echoReply message, the TLV Node ID and the TLV Node Port ID are included in order to share the node and port identifiers with another directly connected device. The remaining TLV types are used in the message format explained below.

In addition, the echoReply message is used by forwarding devices as an acknowledgment to confirm or deny the association in the control tree. In essence, each switch in the network sends an echoReply message not only to announce its topology information but to indicate its association with a specific neighbor (i.e. other switch or SDN controller). To do this, an additional association bit is included in this protocol frame. A description of this association bit is given below.

1) *A*: Association Indicator (1 bit). This one-bit field is used by a network device to announce to its neighbors if it is associated with one of them in the control tree. This bit can turn the echoReply into a join message.

TABLE 2. Summary of TLV supported by eTDP.

Type	TLV Name	Brief Description
0x01	Node ID	Identifies the node that sends the message
0x02	Node Port ID	Provides the node's port identifier
0x03	Neigh ID	Includes the neighbor's unique identifier
0x04	Neigh Port ID	Provides the neighbor's port identifier
0x05	Link Delay	Carries the Round-Trip-Time (RTT) delay to the neighbor

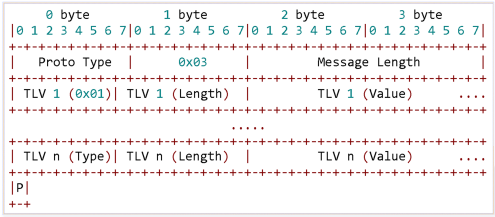

FIGURE 6. topoReply message format.

As a complement, this message enables eTDP nodes to measure RTT latencies in the network. This feature is crucial to support delay-constrained applications or services executed by the control plane properly.

4) topoReply

The principal function of the topoReply message is to guarantee the proper transmission of the topology network state from the forwarding devices to the SDN controllers. To achieve this, this message format is also based on the use of TLV structures.

Fig. 6 shows a brief description of the topoReply message following the basic TLV format. This message may contain the five TLV types supported by the eTDP and presented in Table 2. The first of these (i.e. TLV Node ID), which is mandatory for every topoReply message, identifies the node that sends this message, while the others are used to provide information related to the connectivity with the node's neighbors.

Finally, we have also defined a pruning indicator in the topoReply messages. This pruning indicator is used by the nodes to notify whether they can reach the SDN controllers through only the neighbors receiving this notification. Clearly, nodes that have only one active port (i.e. leaf nodes) and nodes with all its downstream ports pruned (i.e. v-leaf nodes), will send the prune bit set within the topoReply message. A description of this pruning bit is given below.

1) *P*: Pruning Indicator (1 bit). This one-bit field enables eTDP nodes to announce to their neighbors whether they cannot provide an alternative path to the SDN controllers.

The use of the prune bit is intended to provide forwarding devices with a broader knowledge about their local connectivity. At the same time, the prune bit allows switches with

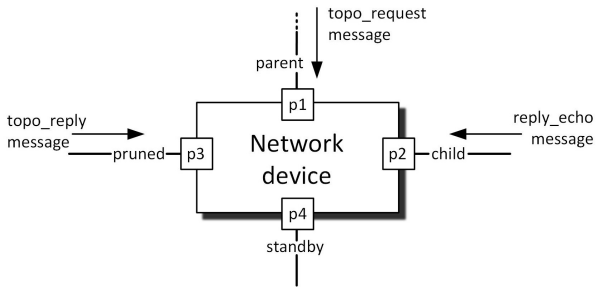

FIGURE 7. Port states for a given network device.

multiple active ports identify themselves as v-leaf or core nodes in the network. This information can be useful for recovery techniques in the face of network failures. Specifically, by using this knowledge switches can reduce the communication overhead during failure notification to the SDN controllers, since prune ports will not be used.

Unlike the two previous messages, the number of fields in the topoReply format is not fixed and depends on the sender position in the resulting control tree.

D. PROTOCOL OPERATION

The presented topology discovery mechanism is initialized by each SDN controller sending a topoRequest message. This multicast message is then propagated accross the network creating a control tree topology rooted at the SDN controllers for collecting network state data. Moreover, this control tree also distributes the management of the physical infrastructure among several SDN controllers.

With the exception of the SDN controller, nodes have one of three roles, i.e. leaf, v-leaf or core, according to their position in the network topology. Leaf nodes are the nodes in the network that have only one neighbor. A node is v-leaf when it has more than one neighbor but only one of them can provide a path to the SDN controllers. The remaining switches are denoted as core nodes.

Additionally, each active port takes one of four states related to the control tree: standby, parent, child or pruned. Fig. 7 shows the port states for a given network device.

- A standby port is an active port in the node that is not used in the control tree.
- A parent port is an upstream port in the control tree that has first received the topoRequest message. Thus, each node has only one parent port.
- A child port is a downstream port of the control tree that has received an echoReply message with the association bit set.
- A pruned port is a child port that has received a topoReply message stating that it is attached to a leaf or v-leaf node.

Each port has a state machine that modifies its current state during the control tree creation. This operation is described in the state transition diagram shown in Fig. 8.

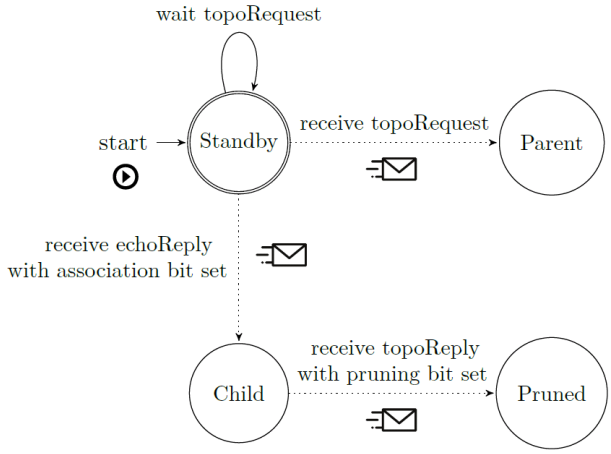

FIGURE 8. Machine state for ports in eTDP.

Algorithm 1 topoRequest Message Forwarding

```
1: Node v receives topoRequest from node u by port p
2: if node v is non-discovered then
3:     Send echoReply to node u                ▷ association bit set
4:     StateMachine (p)                        ▷ p.state = Parent
5:     Send topoRequest for all ports except p
6: else
7:     Send echoReply to node u                ▷ association bit clear
8:     Discard topoRequest
9: end if
```

The state diagram represented in Fig. 8 shows all possible port states drawn as circles. The arcs represent transitions from one state to another and are labeled with the condition that changes the state.

Initially, each network node is in a non-discovered mode, with all its ports in the standby state, waiting for a topoRequest message from an SDN controller or another node. After receiving their first topoRequest message, they become discovered nodes through the proposed eTDP. Algorithm 1 shows the forwarding mechanism for a given node v, after receiving the topoRequest message.

When node v receives a topoRequest from node u, a one-hop echoReply message is sent back to node u. This automatic reply enables the exchange of node and port identifiers between these neighbors as well as the measurement of the RTT in this network link. In addition, the association bit contained in this reply is used by node v as a joining confirmation, to announce to its neighbor node u whether or not they are attached in the control tree.

If when node v receives the topoRequest it is still in the non-discovered state, it confirms the association in the echoReply message, sets the incoming port p to the parent state and forwards the topoRequest for all the remaining ports (with the exception of the incoming port). By contrast, if the topoRequest arrives at an already discovered node (i.e. a node that already has a parent port), it denies the association in

the echoReply and discards the message. Thus, node v has an implicit mechanism that detects and prevents loops.

The proposed topoRequest message forwarding procedure is similar to the FCFS [41] and All-Path [42] approaches. However, in our solution, we have added a one-hop echoReply message. This automatic reply enables important features for the proper functioning of the eTDP mechanism (i.e. the exchange of node and port identifiers between the neighbors, measurement of RTT latencies and the association bit). By adding this message, the switches can perform the topology discovery process and obtain an accurate measurement of the RTT latency simultaneously. The topology information and the delay measure sent by the switches will unlock a more complete and reliable holistic view in the control plane.

As the tree is being created, each switch periodically sends its neighborhood data through the parent port using a topoReply message. The leaf nodes asynchronously start this cyclic process after receiving the topoRequest message. In this case, a one-bit field is added to the topoReply message to change neighboring ports to the pruned state.

Meanwhile, core nodes aggregate topology data from child ports. Once they have received information for all their child interfaces, they complete their topoReply message and send it to the SDN controllers. As a result, topoReply messages are gathered by the SDN controllers, which receive at most an aggregated message from each of their interfaces.

Once the network is discovered the SDN controllers use the resulting control paths to instruct leaf nodes about the retransmission period that must be used. This period must be carefully determined, taking into account the type of network to be discovered. In large-scale geographically distributed networks (i.e. networks that can be considered topologically static), the period value may reach maximum values. By contrast, in cloud or virtualized network deployments (i.e. networks that can be considered dynamic) this period should be adequately analyzed due to existing trade-offs between the number of topology messages forwarded across the network and the required accuracy of the network topology. In addition, given the holistic knowledge of the SDN controllers about the network state, the selected retransmission time can be dynamically adapted according to the network occupation, the applications requirements and the controller's load.

In the face of link failures, switches must inform the controller about port connectivity events. While under OpenFlow specifications, port turned up or down by administrative configurations are notified to the controller using a OFPT_PORT_STATUS message, OpenFlow does not include any mechanism for the switch to inform the controller about link failures or the remote port going down [27]. Precisely, the periodic exchange of topoReply messages defined in eTDP addresses this issue. By using this mechanism, the changes in the network (e.g. link or node failures) are spontaneously notified to the SDN controllers. After receiving this information, the SDN controllers should restart the eTDP mechanism in order to re-establish the affected control tree topology.

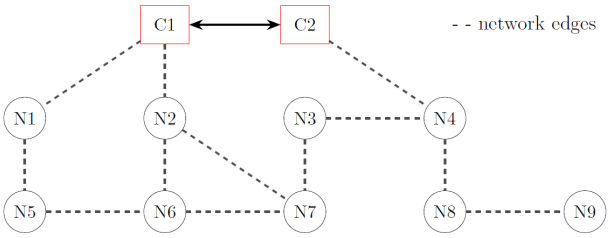

FIGURE 9. Example topology with two controllers and eight switches.

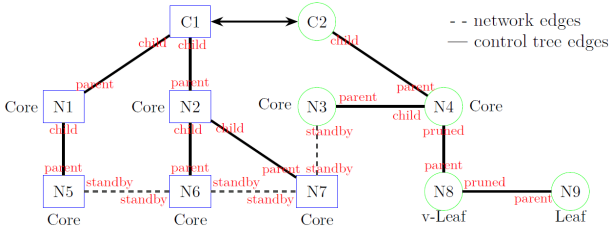

FIGURE 10. Hierarchical control topology of eTDP solution.

Fig. 9 presents an example of the eTDP basic operation. For this sample topology, we consider two SDN controllers connected to eight switches through links that have the same unitary delay. This multi-SDN controller platform can be deployed in the cloud using a resilient, federated architecture [43], [44].

After both SDN controllers run the proposed protocol, the control tree topology and port states are generated, as shown in Fig. 10. The distribution of switches among SDN controllers is depicted in this figure through the use of colors and shapes. A more detailed explanation of how the protocol aggregates the topology data contained in each topoReply message to the SDN controller C1 is provided below.

It can be seen in Fig. 11 the topoReply messages that are sent from the downstream node N5 to the upstream node N1, and then from this upstream node to C1. In general, each node will form its topoReply message putting its locally known topology information first, which includes the node and links to neighbors (i.e. involved ports and delays) from which it has previously received an echoReply. Therefore, the information associated with its parent port (i.e. upstream neighbor and the link between them) will not be included since the node does not have received topology information from its upstream neighbor (Neigh ID and Neigh Port ID). However, this information is provided by the upstream neighbor.

The local topology information provided by each eTDP node is organized in the topoReply message using the following ordered sequence of TLVs: Node ID, Node Port ID, Neigh ID, Neigh Port ID and Link Delay. The last four TLVs are then repeated for every neighbor of the node sending the topoReply message. Then, the node completes its topoReply by aggregating the payloads contained in the topoReply messages received from each of its child ports (if it has such child ports). It should be noted that each additional payload is headed by the use of a new TLV Node ID in the message.

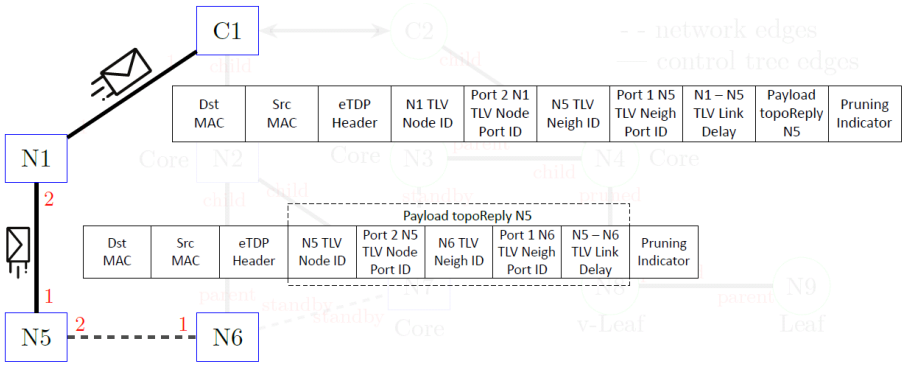

FIGURE 11. Operation example of the proposed eTDP.

Evidently a topoReply from a leaf node will only contain the TLV Node ID.

Using the topoReply received from N1 in Fig. 11, the controller C1 can discover this part of the network topology and determine the resulting control branch. Specifically, the controller will read the ordered sequence of TLVs contained in this message from left to right. In doing this, it will notice that its neighbor, N1, is also connected by its port 2 to port 1 of node N5. It will also discover the delay between these two nodes them. It can also be determined that N5 is connected to the control tree through N1 since the topoReply from N1 includes the payload of the topoReply from N5 (identified by the use of a new TLV Node ID in the message). Lastly, from this payload the controller will also become aware of the connection between N5 and N6. In addition, as the payload from N6 is not aggregated in the topoReply of N5, the controller will know that N6 is not included in the control branch of which N1 and N5 are a part.

The use of the pruning indicator can also be observed in the topoReply messages shown in Fig. 11. These prune bits, all of which are set to zero in the presented example, are sent only one hop back toward the upstream node. Therefore, the upstream node extracts this information from the received topoReply and records it in memory.

1) PROTOCOL COMPLEXITY

The complexity of eTDP is defined in terms of the time and number of messages required to collect the topology information at the SDN controllers and create the hierarchical control tree. Regarding time, the protocol complexity is $O(DT)$, where D is the depth of the control tree and T is the maximum edge delay, which is comprised of the link propagation latency, the transmission delay and the switch processing time. Considering the number of nodes N, the number of controllers C, and the highest number of neighbors per node A, the total number of discovery messages propagated across the network is equivalent to $2[AC+(N-C)(A-1)]+(N-C)$, given that an equal number of topoRequest and echoReply messages are generated and that only one topoReply message is sent by each forwarding device. Therefore, the protocol complexity in terms of messages can be expressed as $O(AN)$.

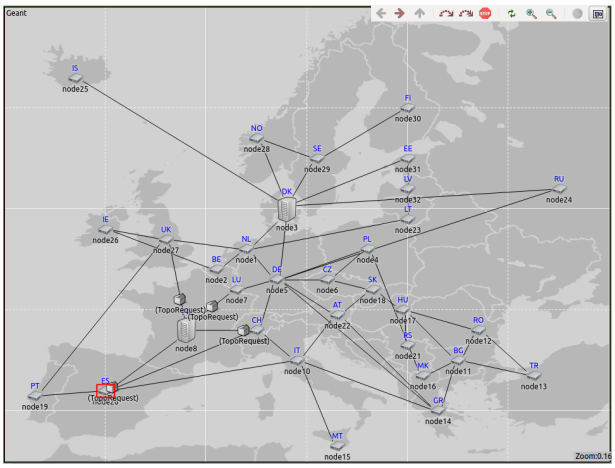

FIGURE 12. Simulation environment in OMNeT++.

V. EVALUATION AND RESULTS DISCUSSION

In this section we evaluate the performance of the proposed topology discovery mechanism. To do this, we use essential metrics to assess the operation of the designed protocol in SDN. Furthermore, we use these metrics to compare the control tree formed by the proposed eTDP against other approaches.

A. SIMULATION ENVIRONMENT

To provide insightful results over a wide range of simulated but realistic scenarios, we have implemented the eTDP mechanism from scratch in the Objective Modular Network Testbed in C++ (OMNeT++) [45]. Fig. 12 shows a sample simulation instance with a European network in the OMNeT++ simulator.

Experimental simulations in OMNeT++ have proven to be a reliable approach for supporting studies in large-scale networks [46] that are not hardware dependent and therefore easy to scale and analyze. The role of experimental simulations using the OMNeT++ network simulator is quite

TABLE 3. Network parameters of the topologies used in the simulations.

Topology	Nodes	Links	Average Degree	Diameter (ms)
Atlanta	15	22	2.93	3.1
Sun	27	51	3.78	3.7
Pioro	40	89	4.45	4.2

significant in the performance evaluation of novel mechanisms in scientific literature [47]. In our study, it enables us to research the behavior and performance of the proposed eTDP in many realistic scenarios of geographically distributed real-world networks.

We generated three sets of networks to evaluate the performance of our solution across varying connectivity degrees. Each network family was generated using one underlying topology from the available online dataset Survivable fixed telecommunication Network Design (SNDlib) [48]. Specifically, we selected three network graphs representative of different scales, namely Atlanta (15 nodes, 22 links), Sun (27 nodes, 51 links) and Pioro (40 nodes, 89 links). Other significant network parameters of these topologies are presented in Table 3.

Topologies that belong to each family set have been constructed as scale-free networks considering a power-law node degree distribution that uses the same degree exponent as that of the original network. This was a result of applying the static Barabási-Albert model [49] and maintaining the original number of nodes and links. Each family size was determined by restricting the margin of error of the indicated average values to less than 6% in each simulation instance. In particular, each topology set is composed of 500 generated networks. All simulation results include a 95% confidence interval based on Student-t distribution.

For each family set different link latencies were randomly generated, considering the mean and standard deviation values of the original network used as the master. For SDN controller placements, we used the most central nodes in each topology based on closeness centrality. In addition, for computing the presented time values, we considered the propagation latencies among nodes in the network and the packet processing time within each node. The switch processing times were determined according to the messages size as given in [50] for NetFPGA implementations.

B. PROTOCOL PERFORMANCE

In this subsection we present the performance evaluation of the eTDP solution for different key metrics and analyze the obtained results.

1) eTDP CONTROL TREE GENERATION

As the eTDP messages propagate across the network, a control tree connecting every forwarding device and rooted at the SDN controller is created. The topology information from each forwarding device is forwarded to the controllers using the generated control tree in order to provide them with a complete network abstract view.

To clearly illustrate the obtained knowledge about the network view and the generated control tree, in Fig. 13 we present a step-by-step explanation of the information received by the controller in the Atlanta topology with a centralized controller as an example. This figure illustrates the reception order of topoReply messages at the controller.

For this evaluation, we placed the controller in the node denoted as N8 and identified with a unique shape (square) and color (blue). The other switches in the network are depicted as black circles. Regarding the links, solid blue lines represent the control connectivity established by eTDP between the network nodes in the resulting tree. Meanwhile, the remaining network links not included in the control tree are rendered as dotted black lines. It should be noted that partially transparent nodes and links represent undiscovered elements of the original topology in the example.

In general, at each step the discovered segments of the entire topology are shown from a holistic point of view in the SDN controller. For instance, Fig. 13(a) reveals the constructed network view after receiving the first topoReply with the corresponding topology data. Specifically, this segment corresponds to the information provided by the topoReply message received from the node denoted as N15, where its connection with N9 is also stated. However, the topology information about N9 is not fully discovered until receiving the second topoReply message with information about the network segment shown in Fig. 13(b). Finally, after receiving the topoReply messages from the other two neighbors (i.e. N3 and N1) the controller's knowledge of the entire network topology is complete, as depicted in Fig. 13(d).

2) DISCOVERY TIME

For the simulations, we have defined the eTDP discovery time as the overall amount of time required by SDN controllers to discover the underlying network topology. This metric measures the period elapsed from the moment when the first topoRequest message is sent to the instant when an SDN controller receives the last topoReply. In Fig. 14, we measure the average discovery time required by a number of SDN controllers to discover the overall network topology.

The proposed discovery mechanism reveals a significant reduction in the average discovery time as the number of SDN controllers increases from 1 to 5. Moreover, this decreasing behavior remains consistent across the three considered topologies with different connectivity degrees. Specifically, reductions of 57%, 43% and 36% are obtained in Atlanta, Sun and Pioro-based families, respectively. This result is expected given that in our protocol, in order to discover the network the SDN controllers must receive an aggregated topoReply message from each of their interfaces with child state in the control tree (i.e. those that received an echoReply with the association bit set). Therefore, with the increase of network controllers a smaller number of nodes are associated

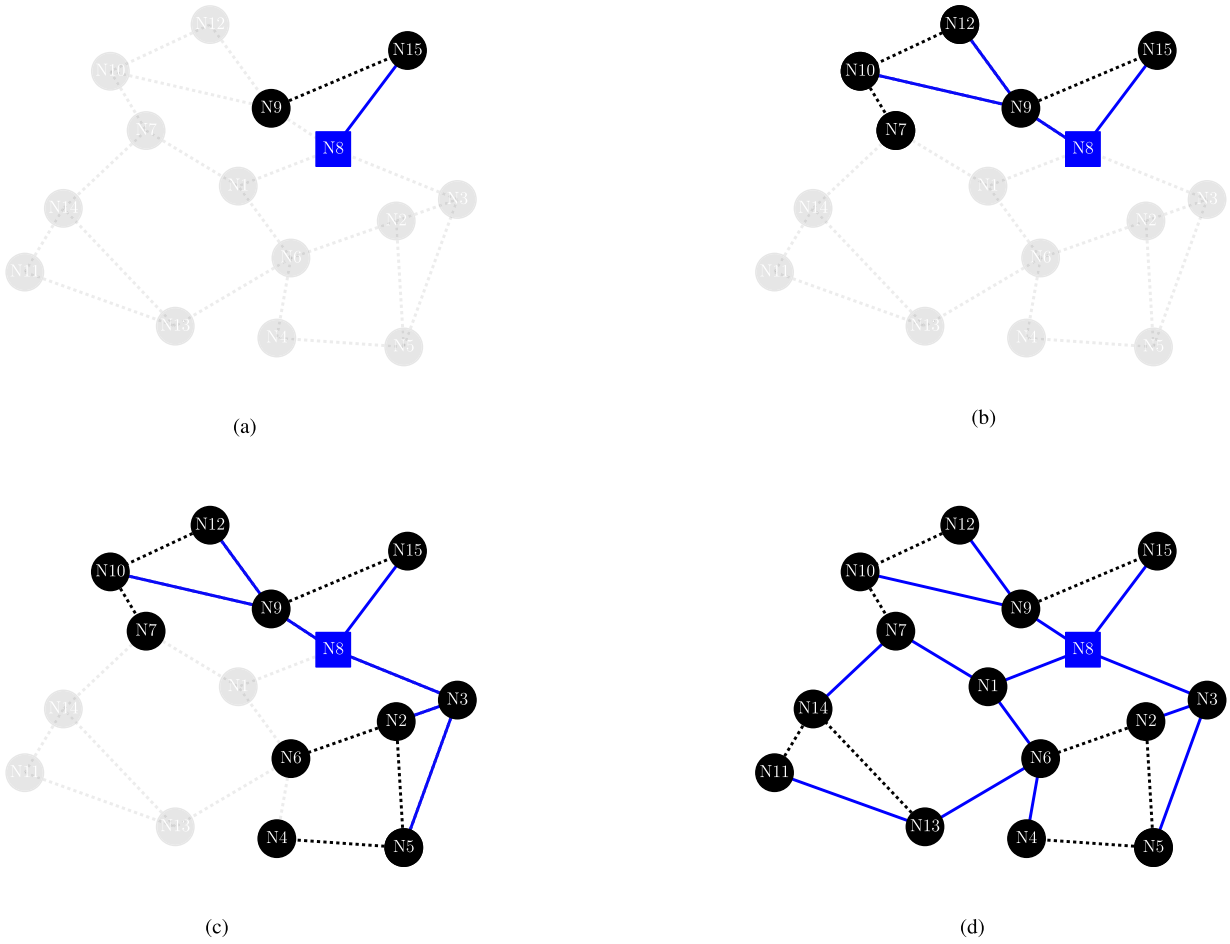

FIGURE 13. Generation of the control tree in atlanta topology. (a) One control branch discovered. (b) Two control branches discovered. (c) Three control branches discovered. (d) Four control branches discovered.

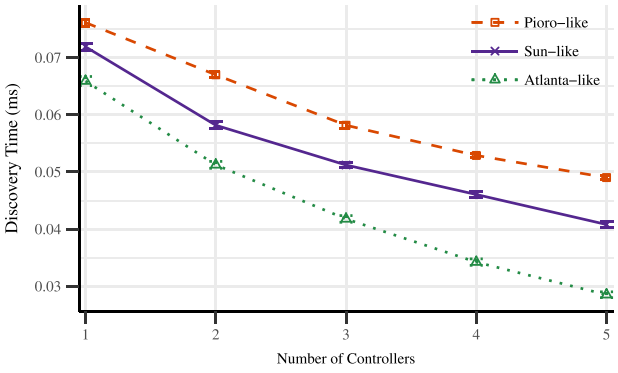

FIGURE 14. Topology discovery time of eTDP.

with each controller (i.e. fewer nodes are included in its control tree), which means that less time is required to obtain the corresponding topology information.

Additionally, in Fig. 14 we can corroborate the relationship between the discovery time and the network diameter (in terms of delay). In essence, network topologies with smaller diameters are more likely to require shorter discovery times, since in our approach control trees are formed using the shortest paths from the controller to each node. Therefore, the associated discovery messages will travel, at the most, the length of the network diameter in their propagation across the network.

3) CONTROLLER OVERHEAD

Measuring the overhead imposed by the topology discovery service on the network controller is of paramount importance in SDN. In this section, we evaluate the number of topology packets the controller sends or receives under different topology discovery mechanisms.

As previously discussed, the topology discovery mechanism implemented by most OpenFlow controllers is called OFDP [20]. The controller load due to OFDP is determined by the number of packet-out and packet-in messages that SDN controllers must process.

Given an OpenFlow-based network of N switches interconnected by L active links, we use p_i to denote the number of ports in a switch $i \in N$. The maximum number of messages sent by the controller for discovering all existing links is defined in Eq. (1), while the number of packet-in messages

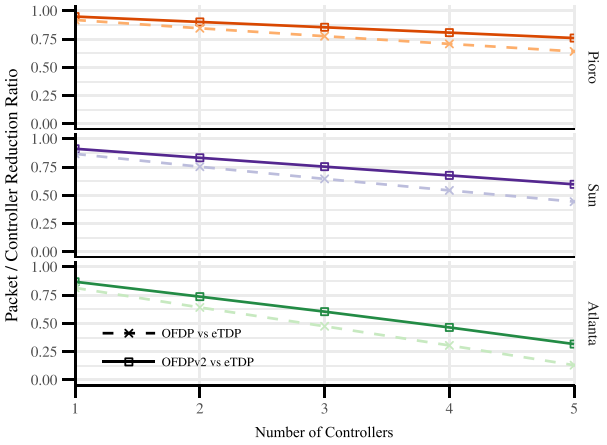

FIGURE 15. Comparison between eTDP and OpenFlow-based protocols.

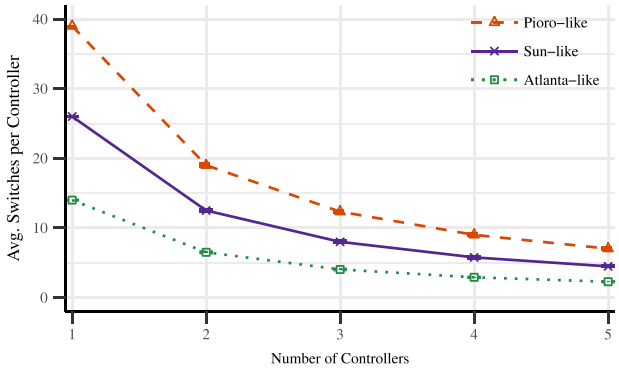

FIGURE 16. eTDP distribution of switches among controllers.

received by the controller is described in Eq. (2).

$$\text{OFDP}_{\text{Packet-out}} = \sum_{i=1}^{N} p_i \tag{1}$$

$$\text{OFDP}_{\text{Packet-in}} = 2 \cdot L \tag{2}$$

In order to achieve a reduction in the number of packet-out messages sent in the OpenFlow-based topology discovery mechanism, an improved version, called OFDPv2, is proposed in [23]. In this second approach, the SDN controller only sends one message per OpenFlow switch, as described in Eq. (3).

$$\text{OFDPv2}_{\text{Packet-out}} = N \tag{3}$$

The reduction of packet-out messages provided by OFDPv2 can be achieved due to the ability of OpenFlow switches to rewrite packet headers. Similar to the previous approach, OFDPv2 takes advantage of the OpenFlow connection establishment between switches and controllers to collect the required topology information. As a result, this improved version consistently performs better than OFDP regarding the number of messages that the controller is required to process. Moreover, OFDPv2 provides a more suitable approach for networks with a higher number of total ports (i.e. networks with higher average switch port density).

In Fig. 15 we present the reduction in the average number of packets managed by the SDN controllers considering two existing approaches as baselines (i.e. OFDP and OFDPv2). This metric is derived from Eq. (4), where *NumPkt* denotes the average number of messages handled (i.e. sent and received) per SDN controller.

$$\frac{NumPkt_{baseline} - NumPkt_{eTDP}}{NumPkt_{baseline}} \tag{4}$$

As shown, in all cases our approach outperforms the two other topology discovery protocols with significant reductions in the number of packets handled per controller that can be over 50%. In general, eTDP achieves noticeable improvements with respect to both baselines, but as expected, larger reductions are achieved in comparison to the use of OFDP. In contrast to OFDP and OFDPv2, in eTDP each controller sends only one packet (i.e. a topoRequest message) and receives at most two packets (i.e. one echoReply and one topoReply) from each of its active interfaces. Consequently, our approach allows decreasing the burden on SDN controllers.

Fig. 15 also illustrates that the improvements with respect to OFDP and OFDPv2 decrease as the number of controllers increases. This behavior is due to the reduction in the number of switches as a result of increasing the size of the controller set. In this case, fewer packets will be required by the two baselines, while in eTDP this number will remain almost constant or will increase (if the new controllers have higher connectivity degrees).

As there is no study that shows OFDP operation for several SDN controllers, in this analysis we assume the switch distribution among SDN controllers obtained from the proposed eTDP. The average number of switches per controller as determined by eTDP is depicted in Fig. 16 for the three considered topology sets.

As expected, the number of switches per controller decreases as the number of controllers grows. Moreover, in the three cases, switches are nearly evenly distributed between controllers at each step along the x-axis. This behavior is the result of the protocol operation in conjunction with the controller's placement assumed in the simulations (i.e. the most central nodes based on the closeness centrality). Under eTDP, switches associate with the controller that is closest to them, forming control trees of shortest paths rooted at the controllers. This reasoning, coupled with the use of controllers placed at the geographic center of the network, results in a balanced load of forwarding devices among the SDN controllers.

4) DISCOVERY PACKETS PER SWITCH

The average number of packets generated per switch to discover the complete network view is shown in Fig. 17. As can be seen, this metric is related to the average network degree.

During the execution of the proposed topology discovery mechanism, eTDP nodes use three main messages

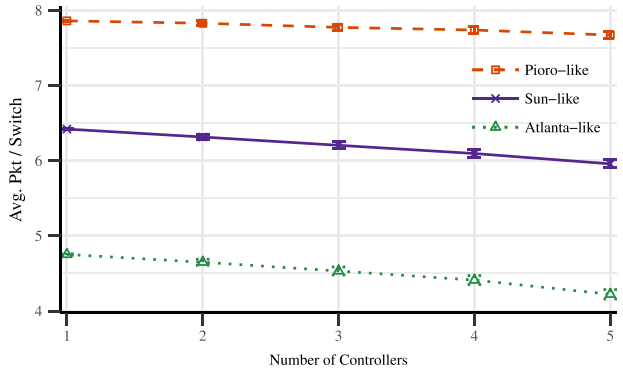

FIGURE 17. Topology discovery packets of eTDP.

(i.e. topoRequest, echoReply and topoReply). However, as each switch always generates only one topoReply message, the average number of packets forwarded in the network is primarily influenced by the topoRequest and echoReply messages. On one hand, the number of topoRequest packets required for discovering the topology corresponds with the number of node neighbors. On the other hand, a switch generates one echoReply per received topoReply. This is equivalent to sending echoReply packets by the parent and standby ports in the forwarding device. As a result, nodes with higher connectivity degrees are likely to generate more topology packets, meaning the resulting average value is therefore increased. The network topologies selected for this study exhibit different connectivity degrees, with the highest values obtained in the Pioro-based topologies.

Furthermore, in all the considered topologies the number of packets handled per switch does not increase in tandem with the number of SDN controllers. Thus, it may be inferred that the eTDP mechanism is scalable in terms of the required number of packets with respect to the number of SDN controllers – a characteristic that is crucial in practical applications.

Table 4 presents the average number of packets generated per forwarding device while the number of SDN controllers is increased in the selected network topologies. These values are also classified according to the types of eTDP control frames.

Under eTDP, echoReply messages exchanged in the network are equivalent to the overall number of generated topoRequest packets, since one echoReply must be generated per received topoReply. However, as shown, more echoReply than topoRequest messages are generated by the switches. This is because the number of topoRequest messages sent by the SDN controllers is not included in this table. Therefore, while an equivalent number of topoRequest and echoReply messages are exchanged between switches, an additional echoReply is generated by those forwarding devices directly connected to the controllers. Evidently, this difference increases with the number of controllers and the number of switches connected to them. Additionally, we can also confirm that the number of topoReply messages generated per switch remains constant and equal to 1, irrespective of the network topology.

TABLE 4. Average number of packets generated per forwarding device.

Classified by Type				
eTDP	**SDN**	**Network Topology**		
Control Frame	**Controllers**	Atlanta	Sun	Pioro
topoRequest	1	1,77	2,60	3,34
	2	1,62	2,46	3,25
	3	1,51	2,35	3,17
	4	1,44	2,26	3,10
	5	1,38	2,17	3,04
echoReply	1	1,99	2,82	3,52
	2	2,03	2,85	3,58
	3	2,04	2,86	3,61
	4	2,06	2,84	3,63
	5	2,06	2,81	3,64
topoReply	1-5	1,00	1,00	1,00
Total	1	4,75	6,42	7,86
	2	4,65	6,32	7,83
	3	4,55	6,21	7,77
	4	4,51	6,10	7,74
	5	4,43	5,98	7,67

TABLE 5. Summary of TLV configuration used in the simulations.

TLV Name	**TLV Length Description** (Bytes)
Node ID	Header (2) + Subtype (1) + Information (1)
Node Port ID	Header (2) + Subtype (1) + Information (1)
Neigh ID	Header (2) + Subtype (1) + Information (1)
Neigh Port ID	Header (2) + Subtype (1) + Information (1)
Link Delay	Header (2) + Subtype (1) + Information (2)

5) eTDP CONTROL TRAFFIC IN THE NETWORK

The eTDP operation is conceived as a cyclical mechanism to be periodically initiated by the leaf nodes. Once the leaf nodes receive a topoRequest message, as previously explained, they send a topoReply message with their topology data through the parent ports. Leaf nodes should continuously initiate this process in order to maintain an accurate global network view in the SDN controller.

In this final evaluation, we capture the overall traffic received by the SDN controller as a result of the eTDP operation after discovering the network topology. To achieve this, we set the retransmission period to 5 *s*, similar to the OFDP implementation in current OpenFlow controllers [6]. In addition, we used the TLV configuration shown in Table 5 to define the length of the required messages.

The different identifiers relating to nodes and ports (i.e. Node ID, Node Port ID, Neigh ID and Neigh Port ID) are represented using alpha-numeric strings of 1 B length. These values are specified in the subtype field as "locally assigned." The value field of the TLV Link Delay is

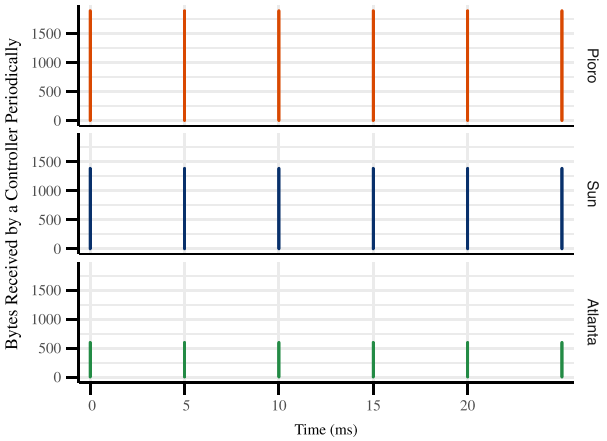

FIGURE 18. Control traffic received at One SDN controller.

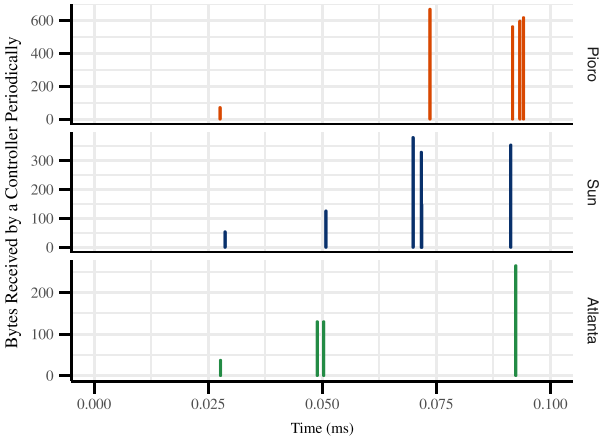

FIGURE 19. eTDP traffic received during one discovery period.

composed of two parts: 1 B of subtype, used in this case to identify the corresponding unit of measure (i.e. s, ms, etc.), and 2 B of information, which contains an integer value between 0 and 65535.

In Fig. 18 we present the traffic periodically received by the centralized controller after discovering the network topology with respect to the three original network topologies.

The purpose of this evaluation is to analyze and compare the control traffic overhead of the controller across the different topologies. Obviously, the volume of received traffic increases as the size of the network grows (in terms of nodes and links) given that more information must be sent to the controller in order to maintain a complete and accurate network view.

Due to the message aggregation strategy employed in the proposed scheme, the topology information periodically sent to the controller is the result of receiving one topoReply message from each controller neighbor. To better illustrate this, Fig. 19 shows the number of bytes received by the controller with the temporal granularity of the plot divided into smaller units.

In this figure it can be seen how several topoReply messages of different sizes successively arrive at the controller. The volume of traffic carried by each message is proportional to the network segment description contained in the data packet unit, and thus to the size of the corresponding branch in the resulting control tree.

VI. CONCLUSIONS

In this paper we proposed a novel protocol for discovering layer 2 infrastructures in large-scale SDN topologies. To that end, the proposed eTDP hierarchically distributes the discovery functions among switches supporting this protocol. Unlike existing approaches, this solution enables automatic discovery of the network without requiring previous IP configurations or controller knowledge of the network. By using this mechanism, the SDN controller is able to discover the network topology and construct a holistic network view without incurring scalability issues while taking advantage of the shortest control paths to each switch. Through experimental simulations with real-world topologies, we have demonstrated that eTDP provides a suitable approach for discovering the network topology with discovery times of under 0.08 ms in the three considered networks. The obtained results also show that the overall number of packets generated per switch is not affected by increasing the number of SDN controllers. Moreover, eTDP achieves noticeable improvements with respect to OpenFlow-based approaches, with the most significant reductions seen in comparison to the current OFDP.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[2] T. Bakhshi, "State of the art and recent research advances in software defined networking," *Wireless Commun. Mobile Comput.*, vol. 2017, no. 4, Jan. 2017, Art. no. 7191647.

[3] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software-defined networking: State of the art and research challenges," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1406.0124

[4] M. Aslan and A. Matrawy, "On the impact of network state collection on the performance of SDN applications," *IEEE Commun. Lett.*, vol. 20, no. 1, pp. 5–8, Jan. 2016.

[5] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, Helsinki, Finland, Aug. 2012, pp. 1–6.

[6] S. Khan, A. Gani, A. A. Wahab, M. Guizani, and M. K. Khan, "Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 303–324, 1st Quart., 2017.

[7] POX. *Network Software Platform*. Accessed: Oct. 25, 2018. [Online]. Available: https://noxrepo.github.io/pox-doc/html/

[8] RYU. *Component-Based Software Defined Networking*. Accessed: Oct. 25, 2018. [Online]. Available: https://osrg.github.io/ryu/

[9] T. Alharbi, M. Portmann, and F. Pakzad, "The (in)security of topology discovery in software defined networks," in *Proc. IEEE 40th Conf. Local Comput. Netw. (LCN)*, Oct. 2015, pp. 502–505.

[10] Global Environment for Network Innovations. *OpenFlow Discovery Protocol and Link Layer Discovery Protocol*. Accessed: Nov. 6, 2018. [Online]. Available: http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol/

[11] *IEEE Standard for Local and Metropolitan Area Networks—Station and Media Access Control Connectivity Discovery*, IEEE Standard 802.1AB-2016 (Revision of IEEE Std 802.1AB-2009), Mar. 2016, pp. 1–146.
[12] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Sydney, NSW, Australia, Jun. 2014, pp. 1–6.
[13] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, Chicago, IL, USA, Aug. 2014, pp. 1–6.
[14] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "Current trends of topology discovery in OpenFlow-based software defined networks," *UPCommons*, Sep. 2015. [Online]. Available: https://upcommons.upc.edu/handle/2117/77672
[15] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw. (Hotnets-IX)*, Monterey, CA, USA, Oct. 2010, pp. 19:1–19:6.
[16] A. Azzouni, N. T. M. Trang, R. Boutaba, and G. Pujolle, "Limitations of OpenFlow topology discovery protocol," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1705.00706
[17] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "Discovering the network topology: An efficient approach for SDN," *Adv. Distrib. Comput. Artif. Intell. J.*, vol. 5, no. 2, pp. 101–108, Nov. 2016.
[18] L. Schiff, S. Schmid, and M. Canini, "Medieval: Towards a self-stabilizing, plug & play, in-band SDN control network," in *Proc. SOSR (Demo)*, 2015, pp. 1–2.
[19] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, "A self-organizing distributed and in-band SDN control plane," in *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, Jun. 2017, pp. 2656–2657.
[20] Open Networking Foundation. (Apr. 2015). *OpenFlow Switch Specification V1.5.1, Technical Specification*. [Online]. Available: https://www.opennetworking.org/
[21] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.
[22] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in software defined networks," in *Proc. 8th Int. Conf. Signal Process. Commun. Syst. (ICSPCS)*, Gold Coast, QLD, Australia, Dec. 2014, pp. 1–8.
[23] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in OpenFlow-based software defined networks," *Comput. Commun.*, vol. 77, pp. 52–61, Mar. 2016.
[24] *Mininet—An Instant Virtual Network on Your Laptop (or Other PC) (Version 2.3.0d4)*. Accessed: Nov. 6, 2018. [Online]. Available: http://mininet.org/
[25] D. Hasan and M. Othman, "Efficient topology discovery in software defined networks: Revisited," *Procedia Comput. Sci.*, vol. 116, pp. 539–547, 2017.
[26] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "TEDP: An enhanced topology discovery service for software-defined networking," *IEEE Commun. Lett.*, vol. 22, no. 8, pp. 1540–1543, Aug. 2018.
[27] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, "sOFTDP: Secure and efficient topology discovery protocol for SDN," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1705.04527
[28] *Floodlight—Open Source Software for Building Software-Defined Networks (Version 1.2)*. Accessed: Oct. 25, 2018. [Online]. Available: http://www.projectfloodlight.org/floodlight/
[29] W.-Y. Huang *et al.*, "Design and implementation of an automatic network topology discovery system for the future Internet across different domains," in *Proc. 26th Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, Fukuoka, Japan, Mar. 2012, pp. 903–908.
[30] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.
[31] G. Tarnaras, E. Haleplidis, and S. Denazis, "SDN and ForCES based optimal network topology discovery," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, London, U.K., Apr. 2015, pp. 1–6.
[32] G. Tarnaras, F. Athanasiou, and S. Denazis, "Efficient topology discovery algorithm for software-defined networks," *IET Netw.*, vol. 6, no. 6, pp. 157–161, Nov. 2017.
[33] Y. Jiménez, C. Cervelló-Pastor, and A. García, "Dynamic resource discovery protocol for software defined networks," *IEEE Commun. Lett.*, vol. 19, no. 5, pp. 743–746, May 2015.
[34] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "Self-healing topology discovery protocol for software-defined networks," *IEEE Commun. Lett.*, vol. 22, no. 5, pp. 1070–1073, May 2018.
[35] J. S. Choi, S. Kang, and Y. Lee, "Design and evaluation of a PCEP-based topology discovery protocol for stateful PCE," *Opt. Switching Netw.*, vol. 26, pp. 39–47, Nov. 2017.
[36] A. Jalili, H. Nazari, S. Namvarasl, and M. Keshtgari, "A comprehensive analysis on control plane deployment in SDN: In-band versus out-of-band solutions," in *Proc. 4th IEEE Int. Conf. Knowl.-Based Eng. Innov. (KBEI)*, Tehran, Iran, Dec. 2017, pp. 1025–1031.
[37] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 3rd Quart., 2014.
[38] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "In-band control, queuing, and failure recovery functionalities for openflow," *IEEE Netw.*, vol. 30, no. 1, pp. 106–112, Jan./Jan. 2016.
[39] R. Callon, *Use of OSI IS-IS for Routing in TCP/IP and Dual Environments*, Standard RFC 1195, Internet Requests Comments, Dec. 1990. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1195.txt
[40] A. DeKok and A. Lior, *Remote Authentication Dial-in User Service (RADIUS) Protocol Extensions*, Standard RFC 6929, Internet Requests Comments, Apr. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6929.txt
[41] K. Miyazaki, K. Nishimura, J. Tanaka, and S. Kotabe, "First-come first-served routing for the data center network: Low latency loop-free routing," in *Proc. World Telecommun. Congr.*, Miyazaki, Japan, Mar. 2012, pp. 1–6.
[42] E. Rojas *et al.*, "All-path bridging: Path exploration protocols for data center and campus networks," *Comput. Netw.*, vol. 79, pp. 862–876, Mar. 2015.
[43] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 4, pp. 436–447, Oct. 2014.
[44] A. C. Risdianto, J.-S. Shin, and J. W. Kim, "Deployment and evaluation of software-defined inter-connections for multi-domain federated SDN-cloud," in *Proc. 11th Int. Conf. Future Internet Technol.*, New York, NY, USA, Jun. 2016, pp. 118–121.
[45] *OMNeT++—Discrete Event Simulator (Version 5.4.1)*. Accessed: Nov. 6, 2018. [Online]. Available: https://www.omnetpp.org/
[46] G. Anggono and T. Moors, "A flow-level extension to OMNeT++ for long simulations of large networks," *IEEE Commun. Lett.*, vol. 21, no. 3, pp. 496–499, Mar. 2017.
[47] A. W. Malik *et al.*, "CloudNetSim++: A GUI based framework for modeling and simulation of data centers in OMNeT++," *IEEE Trans. Services Comput.*, vol. 10, no. 4, pp. 506–519, Jul./Aug. 2017.
[48] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0—Survivable network design library," *Netw., Int. J.*, vol. 55, no. 3, pp. 276–286, May 2010.
[49] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.
[50] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *Proc. 23rd Int. Teletraffic Congr. (ITC)*, San Francisco, CA, USA, Sep. 2011, pp. 1–7.

LEONARDO OCHOA-ADAY received the B.Sc. degree (Hons.) in telecommunication and electronic engineering from the Havana University of Technology "José Antonio Echeverria" and the Ph.D. degree in network engineering from the Universitat Politècnica de Catalunya (UPC). In 2017, he joined the University of Amsterdam, The Netherlands, as a Visiting Researcher. From 2014 to 2015, he was involved in research and development activities with the telecommunications industry. He is currently a Postdoctoral Researcher with UPC. His main research interests include computer networks, the Internet protocols, and network management in software-defined networks, NFV, and 5G.

CRISTINA CERVELLÓ-PASTOR received the M.Sc. and Ph.D. degrees in telecommunication engineering from the Barcelona School of Telecommunications Engineering, Politècnica de Catalunya, Barcelona, Spain, where she is currently an Associate Professor and the Head of the Department of Network Engineering. Being part of the BAMPLA Research Group, she is responsible and actively participated in diverse national and European competitive projects (NOVI, FEDERICA, ATDMA, A@DAN, Euro-NGI, Euro-FGI, and EURONF) and private funding research and development projects. She has published diverse papers in national and international journals and conferences. She is supervising dissertation in the field of management, optimal resource allocation, topology discovery, and routing in software-defined networks/NFV and 5G.

ADRIANA FERNÁNDEZ-FERNÁNDEZ received the Ph.D. degree *(cum laude)* in network engineering from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. In 2017, she joined the System and Network Engineering Research Group, University of Amsterdam, The Netherlands, as a Visiting Researcher. She is currently a Postdoctoral Researcher with UPC. Her research interests include, but are not limited to, communication network modeling and optimization, energy aware routing, traffic engineering, software-defined networks, NFV, and 5G. She received a Scholarship through the Research Training Program (FPI) from the Spanish Government, in 2015.